Automatic conversion of Indian Language Morphological Processors into Grammatical Framework (GF)

Harsha Vardhan Grandhi
LTRC
IIIT Hyderabad
venkata.harshavardhan@research.iiit.ac.in

Soma Paul
LTRC
IIIT Hyderabad
soma@iiit.ac.in

Abstract

Grammatical framework (GF) is an open source software which supports semantic abstraction and linguistic generalization in terms of abstract syntax in a multilingual environment. This makes the software very suitable for automatic multilingual translation using abstract syntax which can be treated as a interlingua. As a first step towards building multi-Indian language translation system using GF platform, we aim to develop an automatic converter which will convert morphological processors available in various formats for Indian languages into GF format. In this paper we develop a deterministic automatic converter that converts LTtoolbox and ILMT morphological processors into GF format. Currently we have converted Hindi, Oriya and Tamil processors using our converter with 100% information preserved in the output. We will also report in this paper our effort of converting Sanskrit and Marathi LTtoolbox morphological processor into GF format.

1 Introduction

Many NLP resources have been developed for processing Indian languages in the last decade. A major consortium of Indian language - Indian language MT system (ILMT) has been successfully carried out for 9 language pairs. These systems mainly follow a transfer based approach. MT system from English to various Indian languages have also been developed through large projects. Morphological analyzers and generators have been developed as part of these projects as well as independently for many Indian languages. In this paper, we present an automatic converter that converts morphological processors that exist in different formats into one common format of Grammatical Framework. Grammatical framework (GF) is an open source software which supports semantic abstraction and linguistic generalization in terms of abstract syntax in a multi-lingual environment (Ranta Aarne, 2009). Abstract syntax can be viewed as an interlingua in the context of multi-lingual machine translation. Apart from abstract syntax there exists another module called concrete syntax in which the morphological specification and syntactic behavior of a language can be captured. The abstract syntax and concrete syntaxes of all languages that one wants to handle together can produce very good quality, especially for domain based MT systems. This is the motivation for converting morphological processors, that at present exist in different format for different Indian languages, into GF format so that good quality meaning driven multi-lingual MT can be accomplished.

Our morphological converter presently converts morph resources designed in Lttoolbox and ILMT morph framework into GF format. We have experimented with Hindi, Oriya, Tamil, Marathi and Sanskrit morphological processors. We have achieved 100% information preservation for the conversion of Hindi, Tamil and Oriya languages. Sanskrit and Marathi LTtoolbox resources are quite huge and we have encountered some issues in conversion which we will discuss in the paper.

The paper is divided into the following sections. In the next section, we will briefly introduce LTtoolbox, the structure of morph used in ILMT and functionalities required for representing morphological information in GF. Section 3 presents related work. Section 4 presents our approach of automatic conversion. In section 5, we will present the result of automatic conversion and analyze the results for Hindi, Oriya (Ithisree Jena and Dipti M. Sharma, 2011) and Sanskrit. We conclude the paper by presenting a critical review of our work

and also insight into future work.

2 Frameworks

We will briefly discuss the architecture of the morphological processors of the three frameworks, Grammatical Framework, Lttoolbox and ILMT.

2.1 Grammatical Framework

GF is a development platform which allows linguists to build grammars (Ranta Aarne, 2009; Ranta Aarne, 2004). It uses paradigm approach in it's morph resource. Parameters and Operations are used as building blocks to create morph resource.

A **parameter** is a user-defined type which is used to model lexical features like number, gender etc.

```
param Number = Sg | Pl
```

Listing 1: " Example parameter "

Here, Number is a parameter representing the singularity/plurality of given word. Each parameter has a set of constructors, which represent possible values of that parameter. In this case, Number has two constructors namely Sg and Pl, which are declared as shown in the example above.

An **Operation** is a function which takes a lemma of the given word and generates a table consisting of all possible word forms. A table in operation consists of branches with constructors on the left and corresponding word forms on the right. Table is computed by pattern matching which returns the value from the first branch whose pattern matches the argument. (Ranta Aarne, 2003)
Ex:

```
oper regNoun : Str -> {s : Number => Str
    } = \dog -> {
        s = table {
                Sg => dog ;
                Pl => dog + "s"
        }
};
```

Listing 2: " Example Operation "

In the example shown above, the operation regNoun takes a String as an argument and returns a Number to String {s: Number => Str }. If the string dog is passed to the operation then it returns a paradigm stating that when the word is singular(Sg) , the operation returns the same word(i.e. dog), if it is plural(Pl) then it returns dogs.

2.2 Lttoolbox

Lttoolbox is an open source finite state toolkit used for lexical processing, morphological analysis and generation of words (Mikel L. Forcada et al., 2011). Like GF, Lttoolbox also uses the paradigm approach for creating morphological analyzer .

Morph resource in Lttoolbox has three important sections (i.e. for automatic conversion) namely symbol definition section, paradigm definition section and lexicon dictionary.

Symbols are used to define lexical categories, features and their values in the morph. These symbols are defined within **symbol definition** (sdef).

```
<sdef n="gen:m" c="masculine" />
```

Listing 3: "Example symbol definition"

In the above example, gen is the symbol for gender, m is the corresponding feature value.

Paradigm takes a lemma and generates all possible word forms. Each paradigm consists of several entries. Each entry has the data to create a word form for a given set of grammatical symbols. These paradigms of the grammar are defined within **paradigm definition** (pardef).

```
<pardef n="kAl/A__adj">
<e><p><l>e</l><r>A<s n="cat:adj"/><s n="
    case:o"/><s n="gen:m"/><s n="num:s
    "/></r></p></e>

</pardef>
```

Listing 4: " Example paradigm definition "

In the above example, the lemma in consideration is kAlA. For the given entry (e) , the word form kAle is generated for kAlA when the grammatical symbols are <cat=adj;case=o;gen=m;num=s>

2.3 ILMT

ILMT uses Computational Paninian Grammar (CPG) for analyzing language and combines it with machine learning. It is developed using both traditional rules-based and dictionary-based algorithms with statistical machine learning (Sampark, 2009).
ILMT morph resource has categories, features, feature_values and word_forms.
Each category with its name and its corresponding features are defined in a single file. Similarly, each feature along with its name and value are stored in a different file. Each category has its own paradigm file. A category can contain many words. Each word belonging to a category

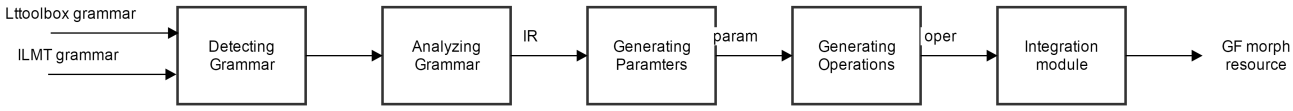

Figure 1: Module level Overview of our approach

is stored in its paradigm file with its root and all its forms. These word forms are listed in last varies first order. For example,

- Noun_m g_m case num means that Noun_m is a category depending on features g_m, case and num.
- case d o represents that case feature has values d and o. Similarly, g_m has value m. num has values s and p.
- Noun_m Gara, Gara, Gara, Gara, GaroM/-Garoz . means that the first word is the root form, and rest are its word forms in last varies fast order of category features(i.e, <m,d,s>, <m,d,p>, <m,o,s>, <m,o,p>).

3 Related Work

There's little work done on automatic resource sharing between frameworks. Some notable works include resource sharing between Apertium and Grammatical Framework (Gregoire and Ranta Aarne, 2014). In this paper, the author proposes an automatic approach to extract Apertium shallow-transfer rules from a GF bilingual grammar. The process of creating GF data from Lttoolbox grammar is done manually. They successfully created and tested the system with English-Spanish language pair. Also some work has been done to import Indian languages (Muhammad Humayoun and Ranta Aarne, 2011) into GF. However, this is manual process and it requires linguists. To the best of our knowledge, there exists no work addressing the automatic conversion from existing morph resources(ILMT/Lttoolbox) of Indian languages into GF.

4 Our Approach

In this section, we describe our approach of converting LTtoolbox and ILMT morph resources into GF morph resources. The steps of conversion is presented in the figure 1.

Our approach converts the morphological processors available either in XML or ILMT syntax format and converts them into Grammatical Framework. The key idea here is that Grammatical Framework is a programming language with a definite syntax and we need to transform the given morphological processors such that they adhere to Grammatical Framework's syntactic structure. This calls for the need to change morphological processors into non-ambiguous, permissible units which are supported by Grammatical Framework. Some examples of above mentioned challenges are as follows. Further details regarding these decisions are described in algorithms mentioned below.

- As already mentioned in above sections, Lttoolbox supports paradigms which depend on variable number of parameters whereas Grammatical Framework only supports dependence on pre-determined parameters. So we need to analyze all paradigms and their dependencies to find the specific occurrences and transform them into tabular syntax structure, supported by Grammatical Framework.

- Some attributes present in the source files (e.g. numeric attributes, duplicate attributes) cannot be transformed directly into GF e.g. parsarg = 0. We chose to modify the parameters to include the actual parameter value and other contextual information e.g. the parameter type, the paradigm name etc.

4.1 Detecting Grammar

In this step, we classify the user input as Lttoolbox or ILMT morph resource. This is a relatively simple step because of widely different syntaxes of both morph resources. We accomplish this using syntax-based heuristics.

4.2 Analyzing Grammar

In this step, we take the respective grammar, parse it and then convert into an intermediate representation (IR). The motivation behind this step is to reduce complexity for the following steps i.e Generating parameters and operations.
In the case of ILMT, the morph resource is present

across different sources. So we convert it into XML using the same structure as in Lttoolbox, to make the conversion process more generalized. Once we have the XML source, we will convert it into IR by using the algorithm shown in Algorithm.1.

Algorithm 1 " Algorithm for Analyze Grammar "

features = {} ▷ sdef - represents the set of all symbol definitions
for s ∈ sdef **do** ▷ s contains lexical features and lexical values
    features[s.lexical-feature].add(s.lexical-value)
paradigms = {} ▷ pardef - represents the set of paradigm definitions
for p ∈ pardef **do**
    root = p(n) ▷ root word
    for e ∈ p **do** ▷ e is an entry in paradigm definition ▷ e contains feature set and word form
        paradigms[root].add(e.feature-set,e.word-form)
return features,paradigms

In algorithm 1, we are converting symbol definitions and paradigm definitions into IR (i.e features and paradigms).

4.3 Generating Parameters

In this step, we use the features from IR to generate parameters and their constructors, using the algorithm shown in Algorithm 2. The function

Algorithm 2 " Algorithm for generating parameters "

for f ∈ features **do** ▷ features from IR ▷ f contains feature names and values
    buildParameters(f.name, f.value)

buildParameters generates GF syntax (parameters and corresponding constructors) for a given feature name and its values. For example calling the function buildParameters with arguments (Num,[num_s,num_p]) returns

```
Num = num_s | num_p;
```

Listing 5: " Example Parameter Syntax in GF "

4.4 Generating Operations

In this step, we use the IR (features, paradigms) to generate operations using the algorithm given in

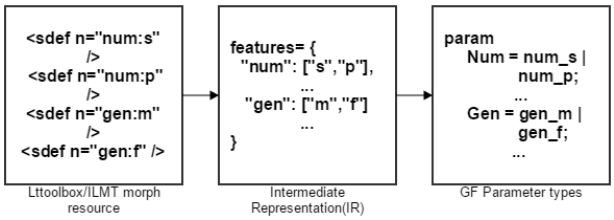

Figure 2: The entire process of generating parameters 1.Lttoolbox/ILMT morph as given by user 2.IR created in the previous step 3.GF morph resource parameters created in the present module.

Algorithm 3 " Algorithm for generating operations "

function BUILDOPERATIONS(features, paradigms)
    for p ∈ paradigms **do**
        buildDecalaration(p,dependentFeatures)
        buildTable(features,p,dependentFeatures)
    return

Algorithm 3. The function buildOperations builds operations for each paradigm. As explained in the above section, each operation consists of declaration and description which are generated by buildDeclaration and buildTable respectively. To do this we need to keep track of all the features used for a paradigm. So we built a helper function which does this by iterating through all the entries in the paradigm.

For example, calling buildDeclaration with arguments (case,gen,num) produces the x1,,,,,,x8:Str -> Case => Gen => Num => Str (we used x1-x8 because each feature has two values, total possible values in table are 2x2x2 = 8).

The function buildTable recursively builds the table which is used to generate the possible word forms for a given lemma in GF, using the algorithm shown in Algorithm 4.

4.5 Code and Datasets

The entire code base has been written in Python. The description of datasets are shown in Table 1. Code and datasets are publicly available under open-source license.[1].

[1] https://github.com/harshavardhangsv/automaticMorphResourceConverter

Algorithm 4 " Algorithm for building table"

function BUILDTABLE(features,paradigm, dependentfeatures)
  if dependentFeatures is empty **then**
    return
  for df ∈ dependentFeatures **do**
    for v ∈ features[df] **do**
      v => buildTable(features, paradigm, rest(dependentFeatures))
                                ▷ rest(1,2,3) = 2,3

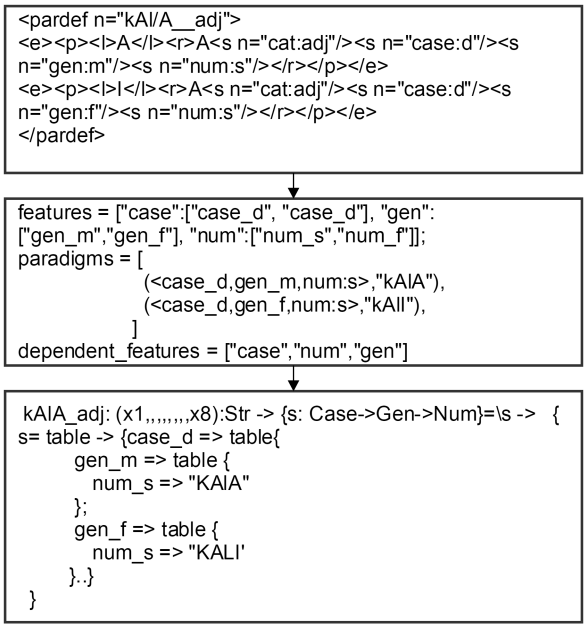

Figure 3: This figure explains the process of generating operations. 1.Lttool/ILMT morph resource given by user 2.variables state in IR form 3.GF syntax operations created from the buildTable

Dataset	# of paradigms	# of words
Hindi	57	26356
Tamil	254	34290
Oriya	45	2860
Bengali	148	5478

Table 1: Description of Datasets

5 Evaluation and Error Analysis

We measure the accuracy of the converter in the following way. The same text is given to both GF analyzer and the original source tool (LTtoolbox and ILMT). We verify that the output of GF analyzer and the original source tool is identical. Thus we ensure that information is faithfully transferred. We found that there is no loss of data for the chosen languages i.e Hindi, Oriya, Tamil and Bengali. This evaluation is repeated for various random news articles and 100% information was preserved in the output. However, we have come across with some problems in converting Sanskrit and Marathi morph resources. These problems are mainly caused due to presence of language-specific syntax which we had trouble in generalizing. Even though we had some troubles, we have found that, after analyzing files manually, our approach will still be able to make the automated conversion to around 65%.

6 Conclusion and Future Work

In this paper, we have explained the process of creating an automatic converter which deterministically converts morphological processor from different formats into GF format without any information loss for Hindi, Oriya and Tamil. We are presently working on other Indian languages such as Marathi, Sanskrit and Telugu. We have observed that if there exists any language specific morpho-syntactic information in the resource, conversion of such resources requires additional work. At present we are converting morph resources created in Lttoolbox and ILMT morph format.

In future we might come across a morphological processor developed in another framework. In order to bring the complete genericness into our tool, we want to modularize our system. We intend to develop a generic system that first asks the user about the format of their morphological processor and then activate the right module for the conversion. We hope that with our effort we will be able to offer a NLP resource to the community which will eradicate the barrier of framework differences.

References

Gregoire Dietrez and Ranta Aarne. 2014. *Sharing resources between free/open-source rule-based ma-*

chine translation systems: Grammatical Framework and Apertium. LREC.

Ithisree Jena, Dipti M. Sharma 2011. *Developing Oriya Morphological Analyzer Using Lt-Toolbox* . Communications in Computer and Information Science pp.124-129.

Mikel L. Forcada, Mireia Ginest-Rosell, Jacob Nordfalk, Jim ORegan, Sergio Ortiz-Rojas, Juan Antonio Prez-Ortiz, Felipe Snchez-Martnez, Gema Ramrez-Snchez and Francis M. Tyers 2011. *Apertium: a free/open-source platform for rule-based machine translation* . In Machine Translation: Volume 25, Issue 2 (2011), p. 127-144.

Muhammad Humayoun , Ranta Aarne 2011. *Developing Punjabi Morphology, Corpus and Lexicon* . Proceedings of the 24th Pacific Asia Conference on Language, Information and Computation.

Ranta Aarne 2003. *A revised version of the on-line GF tutorial, v1.0.*. In A. Beckmann and N. Preining, editors, ESSLLI Course Material I (2003).

Ranta Aarne 2004. *Grammatical Framework: A Type-Theoretical Grammar Formalism*. The Journal of Functional Programming 14(2) (2004) 145189.

Ranta Aarne 2009. *The GF Resource Grammar Library: A systematic presentation of the library from the linguistic point of view* . Linguistics in Language Technology, 2(2).

Ranta Aarne 2009. *Grammars as Software Libraries*. From Semantics to Computer Science, Cambridge University Press, Cambridge, pp. 281- 308.

Sampark: Machine Translation System among Indian languages 2009. http://tdildc.in/index.php?option=com_vertical &parentid=74, http://sampark.iiit.ac.in/.